CSE127, Computer Security

System Security I: Secure Design, Processes, Kernels, VMs

UC San Diego

Housekeeping

General course things to know

- PA2 is released, due at **1/29** at 11:59
  - Note a two day preextension — this one is hard and my course staff suggested a little extra time might be appreciated
  - Discussion will give good hints and tips
  - Get started early (you can’t really grind this one, you have to think a lot)

Previously on CSE127…

Application security

- So far we've learned lots of ways to **corrupt control flow**
  - Stack overflow, pointer subterfuge, format strings, etc.
- Once you corrupt control flow, attacker can **run code of their choice**
  - Either directly (i.e., shellcode in a buffer) or using return-oriented programming
- Mitigations can make this harder, but don't fully stop this
- So… you might be feeling a little despondent…

Today's lecture — Systems Security

Learning Objectives

- Understand definitions of privilege, privilege separation, defense in depth, and why we need these in computer systems
- Think about *abstractions* as *trust boundaries* and apply these to memory, processes, and even OSes themselves
- Learn the basics of VMs and virtualization, and the guarantees of virtual machines

Secure Design Principles

A hypothetical

Some bad C code

```
int main() {
    char *p = NULL;
    *p = 20;
}
```

- What does this code do?
- What do you think will happen when this code runs?
- Do you expect your entire system to crash if you run this code?

A hypothetical

Some bad C code

```
int main() {
    char *p = NULL;
    *p = 20;
}
```

- **What does this code do?**
- What do you think will happen when this code runs?
- Do you expect your entire system to crash if you run this code?

A hypothetical

Some bad C code

```
int main() {
    char *p = NULL;
    *p = 20;
}
```

- What does this code do?
- **What do you think will happen when this code runs?**
- Do you expect your entire system to crash if you run this code?

A hypothetical

Some bad C code

```
int main() {
    char *p = NULL;
    *p = 20;
}
```

- What does this code do?
- What do you think will happen when this code runs?
- **Do you expect your entire system to crash if you run this code?**

Does the whole system crash on a NULL pointer reference?

- No! At least, not on modern systems…
- But back in the 80s/90s… absolutely!
  - MS-DOS / IBM DOS, NULL ref will **crash the whole system**
  - Why?

Does the whole system crash on a NULL pointer reference?

- No! At least, not on modern systems…
- But back in the 80s/90s… absolutely!
  - MS-DOS / IBM DOS, NULL ref will **crash the whole system**
  - Why?
    - *No* protection or isolation from the underlying system
    - *No* memory protection (all memory is available to access to all processes…)
    - *No* protected OS kernel (all processor operations available to programs)

Secure System Design Principles

- With a group, come up with definitions and examples to the following concepts:
  - Least privilege?
  - Privilege separation?
  - Complete mediation?
  - Defense-in-depth?

Secure System Design Principles

- With a group, come up with definitions and examples to the following concepts:
  - **Least privilege?**
  - Privilege separation?
  - Complete mediation?
  - Defense-in-depth?

Principle of Least Privilege

- Simple definition: Only provide as much privilege to a program (or entity, person, etc.) as is *needed* to its job

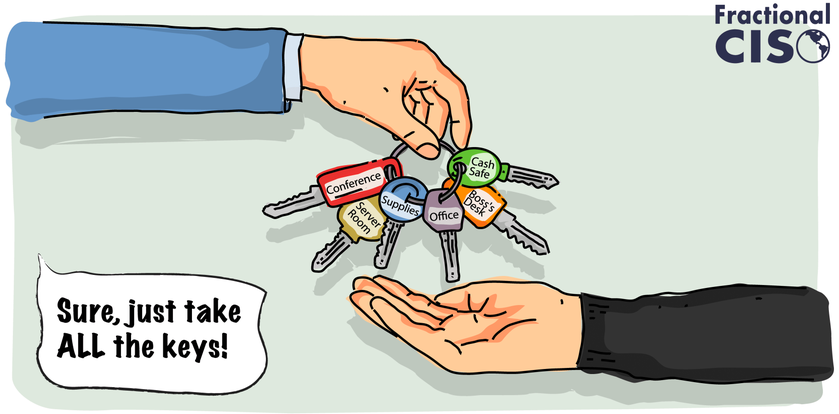
- What assumptions does this definition make?

Principle of Least Privilege

- Simple definition: Only provide as much privilege to a program (or entity, person, etc.) as is *needed* to its job
- What assumptions does this definition make?
  - The job must be **clearly defined**
- Aka… no functions that do 28 things
- What are some examples of least privilege?

Principle of Least Privilege

- Simple definition: Only provide as much privilege to a program (or entity, person, etc.) as is *needed* to its job
- What assumptions does this definition make?
  - The job must be **clearly defined**
- Aka… no functions that do 28 things
- What are some examples of least privilege?
  - Non-root accounts can't install programs
  - Students can *view* Gradescope but can't modify Gradescope



not least privilege

https://fractionalciso.com/least-privilege-is-key-to-good-cybersecurity/

Secure System Design Principles

- With a group, come up with definitions and examples to the following concepts:
  - Least privilege?
  - **Privilege separation?**
  - Complete mediation?
  - Defense-in-depth?

Privilege Separation

- Simple definition: Divide system into different pieces, each with separate privileges, requiring multiple different privileges to access sensitive data / code

- What are some examples of privilege separation?

Privilege Separation

- Simple definition: Divide system into different pieces, each with separate privileges, requiring multiple different privileges to access sensitive data / code

- What are some examples of privilege separation?

  - For a website that uses passwords — main server handles requests (so can interface with users), passes data to a *password* server for authentication (so can only interface with the database)

- Fundamental type of attack: **privilege escalation**; where an attacker can get higher privileges than they are allocated

Secure System Design Principles

- With a group, come up with definitions and examples to the following concepts:
  - Least privilege?
  - Privilege separation?
  - **Complete mediation?**
  - Defense-in-depth?

Complete Mediation

- Simple definition: Check **every** access that crosses a trust boundary against a security policy
  - Assumes a well defined and *checkable* security policy!
- What are some examples of complete mediation?

Complete Mediation

- Simple definition: Check **every** access that crosses a trust boundary against a security policy
  - Assumes a well defined and *checkable* security policy!
- What are some examples of complete mediation?
  - Bouncers (why?)
  - TSA
  - Memory accesses (check permissions on *every* read/write, not just when you load a program)

https://www.usadojo.com/guardians-of-the-night-bouncers/

Secure System Design Principles

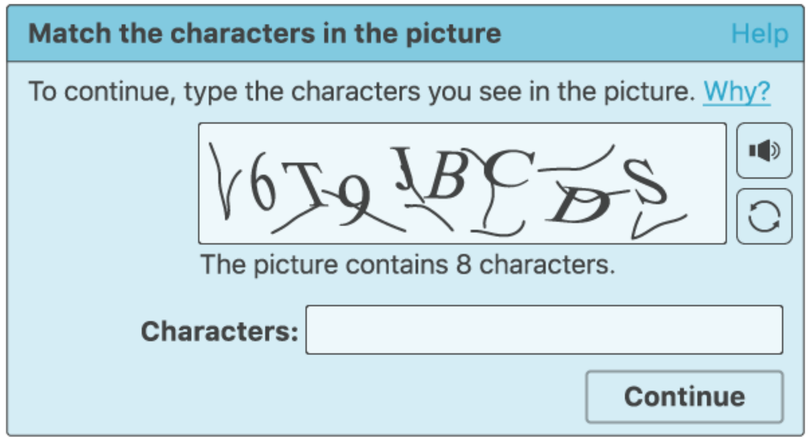
- With a group, come up with definitions and examples to the following concepts:
  - Least privilege?
  - Privilege separation?
  - Complete mediation?
  - **Defense-in-depth?**

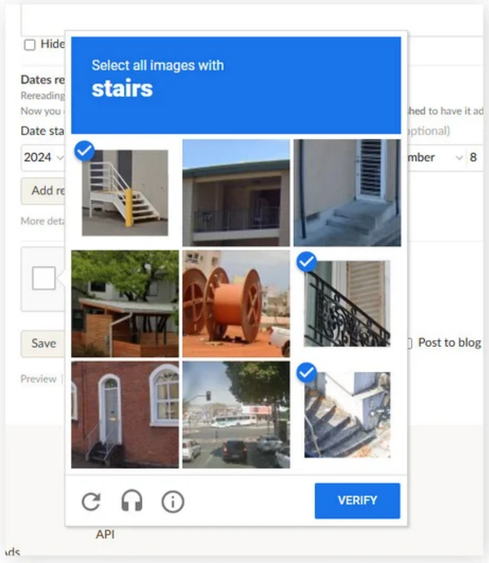
Defense in Depth

- Simple definition: Use more than one security mechanism for protection.
- What are some examples of defense in depth?

Defense in Depth

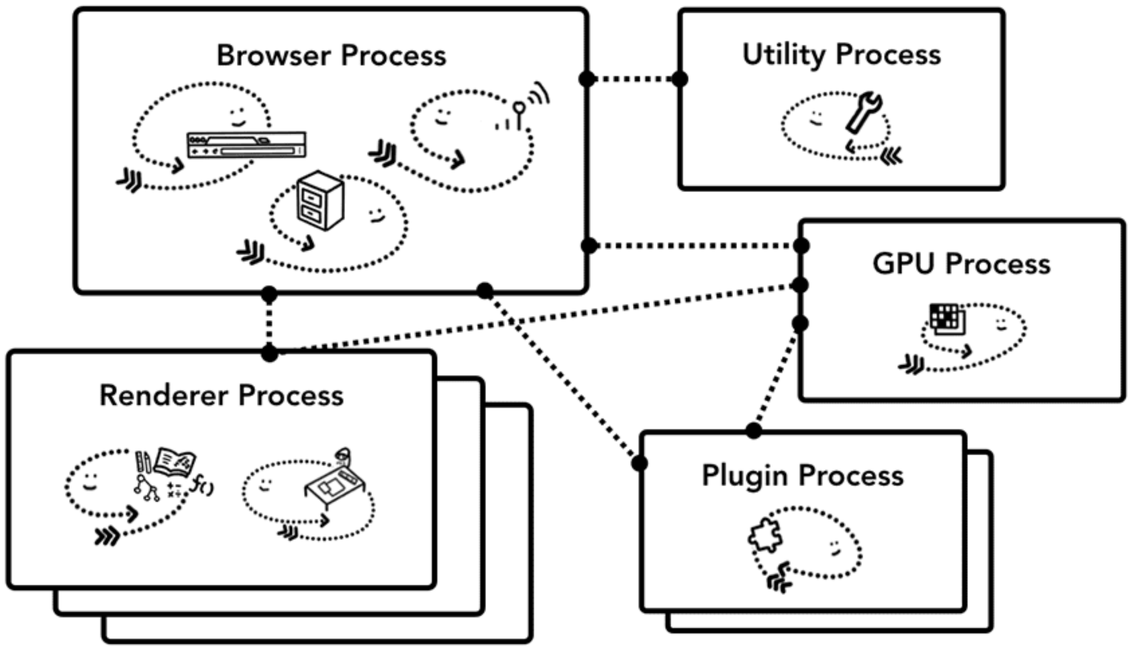
- Simple definition: Use more than one security mechanism for protection.
- What are some examples of defense in depth?
  - Bridges *and* moats to protect the castle
  - Passwords *and* CAPTCHAs (why?)





Concrete Example: Web Browser architecture

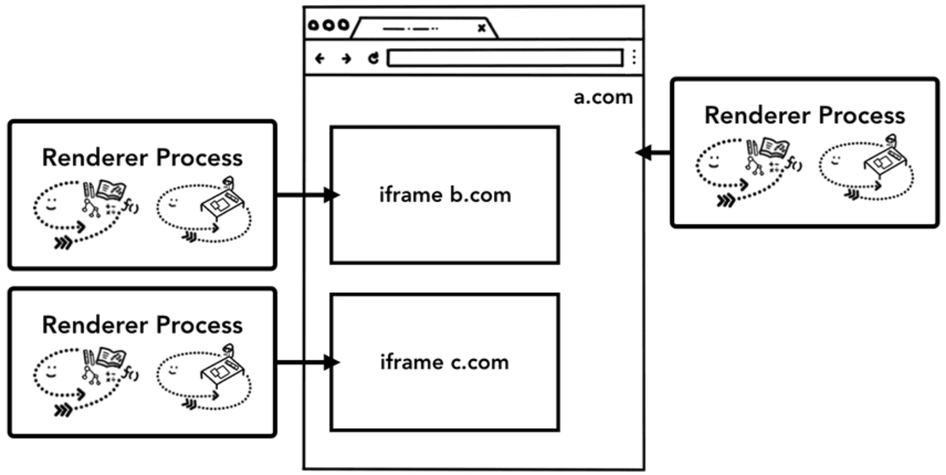
- Different processes control different components (aka least privilege + privilege separation)
  - Browser process controls address bar, bookmarks, back / forward buttons
  - Renderer controls anything where a website is displayed
  - Plugin processes control extensions, etc.
  - GPU process handles code strictly for GPU (and nothing else)



https://developer.chrome.com/blog/inside-browser-part1

Concrete Example: Web Browser architecture

- Chrome offers **site isolation**
  - Run a separate renderer processes for each tab / website
  - Run a separate renderer process for each **frame** inside of a webpage
    - We'll talk more about frames in the web unit!



https://developer.chrome.com/blog/inside-browser-part1

How does this work in modern OSes?

- What are some design principles that offer security in modern operating systems?

How does this work in modern OSes?

- What are some design principles that offer security in modern operating systems?
  - **Process abstraction**
    - Processes have user UIDs that determine what they're allowed to access on the system
  - **Process isolation**
    - Processes can't interfere with other processes memory (aka, buffer overflow in one program doesn't harm another program)
  - **User/Kernel isolation**
    - Privileged operations happen in the kernel
    - User requests are checked by the kernel against some security policy

Process Trust Boundaries

Process Isolation

- The process abstraction is one of isolation; processes are not by default allowed to talk to one another
  - The process boundary is a trust boundary
  - Any inter-process interface is part of the attack surface (including reading / writing from files!)
- How are individual processes isolated from one another?

Process Isolation

- The process abstraction is one of isolation; processes are not by default allowed to talk to one another
  - The process boundary is a trust boundary
  - Any inter-process interface is part of the attack surface (including reading / writing from files!)
- How are individual processes isolated from one another?
  - File access control lists
  - Virtual memory
- What do we trust to manage the isolation of individual processes?

Process Isolation

- The process abstraction is one of isolation; processes are not by default allowed to talk to one another
  - The process boundary is a trust boundary
  - Any inter-process interface is part of the attack surface (including reading / writing from files!)
- How are individual processes isolated from one another?
  - File ACLs
  - Virtual memory
- What do we trust to manage the isolation of individual processes?
  - The OS!

UNIX File System

- What does “`ls`” actually show you?

```
$ ls -l
drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

UNIX File System

- What does "`ls`" actually show you?

```
$ ls -l
drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- What are these?

UNIX File System

- What does "`ls`" actually show you?

```
$ ls -l
drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- What are these?
  - user who owns the file (with a UID) and group who owns the file (with a GID)

UNIX File System

- What does "`ls`" actually show you?

```
$ ls -l
drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- What are these?
  - user who owns the file (with a UID) and group who owns the file (with a GID)
- How do you parse this string?

UNIX File System

- What does "`ls`" actually show you?

```
$ ls -l
drwxr-xr-x. 4 root root    68 Jun 13 20:25 tuned
-rw-r--r--. 1 root root  4017 Feb 24  2022 vimrc
```

- What are these?
  - user who owns the file (with a UID) and group who owns the file (with a GID)
- How do you parse this string?
  - First character is file type (- for files, d for directory)
  - Next is a group of three sets of permissions: owner (rwx), group (rwx) and other (rwx)

Understanding octal values

- Octal values tell you the permission levels for files, summed across permissions
  - r (read) 4; w (write) 2; x (execute) 1
- So a permission value of 764 means…
  - First 7; user who owns file can read/write/execute
  - Second 6; group who owns the file can read/write (not execute)
  - Last 4; anyone can read the file
- Change permissions with **chmod** (change mode); e.g.,
  - `chmod 777 <filename>`

How do processes have access to files?

- Permissions in UNIX are granted according to UID
  - UID is set by parent process (e.g., if **kumarde** runs the shell which runs the program, the program's UID would be **kumarde**)
  - Special user **root** has UID 0… they can access *any* file
- Each file (as we saw earlier) has an *Access Control List* (ACL); basically the permissions string that allow programs to read other files
  - Side note… everything is a file. Everything**.** Once you realize that almost all of CS is just opening and closing files… you're finally ready to graduate

How do processes have access to files?

- OK, but… then how does my computer work?
  - Consider changing your password using `passwd`
  - `passwd` needs to modify `/etc/passwd`, which is a file owned by root… so how do regular users just use it?

How do processes have access to files?

- OK, but… then how does my computer work?
  - Consider changing your password using `passwd`
  - `passwd` needs to modify `/etc/passwd`, which is a file owned by root… so how do regular users just use it?

```
-rwsr-xr-x 1 root root 63K May 30  2024 /usr/bin/passwd
```

How do processes have access to files?

- OK, but… then how does my computer work?
  - Consider changing your password using `passwd`
  - `passwd` needs to modify `/etc/passwd`, which is a file owned by root… so how do regular users just use it?

  - Enter: **setuid** bit
    - A program can have a bit called setuid in its permissions
    - Each process has three UIDs: real user ID (rUID), effective user ID (eUID), saved user ID (sUID)
    - If so, caller's EUID is set to the UID of the file (which is a temporary privilege escalation!)… super dangerous, but can be safe if you're smart

Virtual Memory

- What is virtual memory?

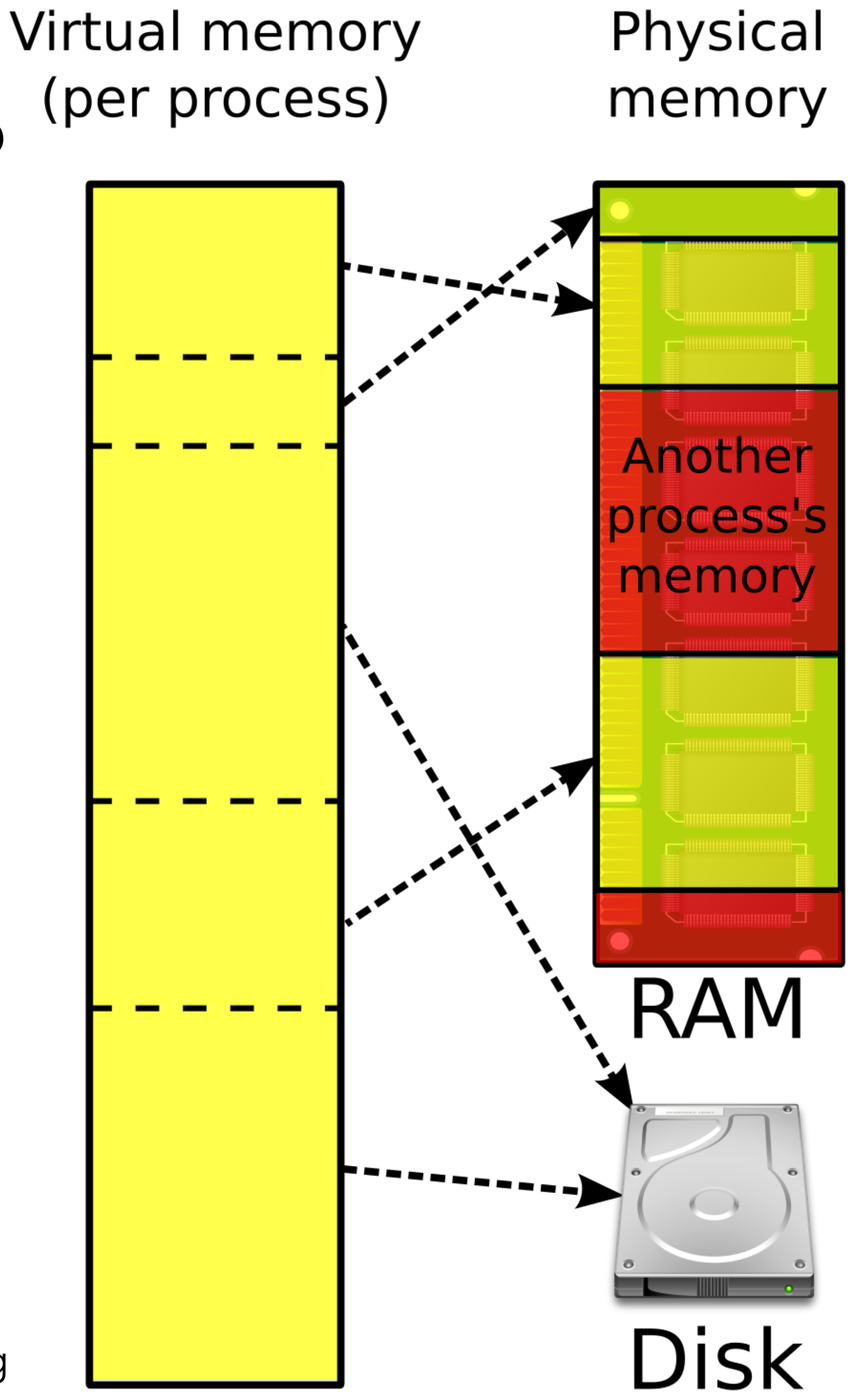
Virtual Memory

- What is virtual memory?
  - Each process gets its own “virtual address space,” completely managed by the operating system
  - Each process thinks it has access to the entire memory space, as if this is the only process running on the system
- This is a beautiful way to both **multiplex** the resources of the OS (probably how you learned about this in 120) but also enforce **security boundaries** between processes
  - Two birds

https://c1.staticflickr.com/2/1603/26666393696_48c102e12f_b.jpg

How does virtual memory work?

- Memory addresses used by processes are *virtual addresses*
- Virtual addresses are mapped by the operating system into *physical addresses*, corresponding to actual storage locations
- The OS does this with an MMU (memory management unit) on the CPU to conduct *address translation* (the mechanism to map virtual to physical addresses)



https://en.wikipedia.org/wiki/Virtual_memory#/media/File:Virtual_memory.svg

Ok… but in practice, really, how is it done?

- Page
  - Smallest unit of data for memory management in an OS using virtual memory
  - Usually 4KB (or multiple thereof) (12-bits)
- Translations happen through a **page table, one per process**
  - Keeps track of mapping between virtual memory for a process and physical memory address
  - Fully mapping all possible virtual addresses to all possible physical addresses is impossible (would require exabytes of storage); so we use a **multilevel page table**

Address Translation

- Every memory access a process performs goes through address translation (mostly)
  - Load, store, instruction fetch
  - In that sense, the MMU does “complete mediation” of memory accesses
- That is **super** expensive!
  - What kinds of data structure can help us not look stuff up that we might already know about?

Address Translation

- Every memory access a process performs goes through address translation (mostly)
  - Load, store, instruction fetch
  - In that sense, the MMU does “complete mediation” of memory accesses
- That is **super** expensive!
  - What kinds of data structure can help us not look stuff up that we might already know about?
  - A cache! In this case, the **translation lookaside buffer (TLB);**

Translation Lookaside Buffer

- Small cache of recently translated page addresses (in hardware)
  - Before translating a referenced address, the processor checks the TLB
- Identifies
  - Physical page corresponding to the virtual page (or if the page is not in memory at all)
  - If page mapping allows the ***mode of access*** (e.g., r/w/x), then allows whatever the process wants to do (aka, enforces access control)

Wait… not everything is accessible to the process?

- Of course not! Recall DEP? W^X? How does that actually work in practice?
  - Page descriptor (in the page table) contains additional access control information
    - read, write, execute permissions
    - These are usually low-order bits in the page table entry
- Set by the operating system and/or user programs (mprotect())
- If a program attempts the wrong mode of access, the processor will generate a **fault** and tell the OS to handle it

In sum, for processes

- Virtual memory offers clear delineation between process boundaries
- Any interprocess communication is dangerous (and should be highly considered when developing secure code)
- All of this is handled by the **OS;** not the process itself — this is a fundamental design decision by developers

OS Trust Boundaries

Operating System Protections

- Now you're protecting the operating system itself. What are the assets you're trying to protect?

Operating System Protections

- Now you're protecting the operating system itself. What are the assets you're trying to protect?
  - Secret memory (e.g., passwords)
  - Ability to run arbitrary programs
  - Ability to download and install programs
- What's your attack surface?

Operating System Protections

- Now you're protecting the operating system itself. What are the assets you're trying to protect?
  - Secret memory (e.g., passwords)
  - Ability to run arbitrary programs
  - Ability to download and install programs
- What's your attack surface?
  - Memory accesses, privileged instructions, system calls and faults, device accesses… anywhere you're not talking to **OS driven code**

Operating System Protections

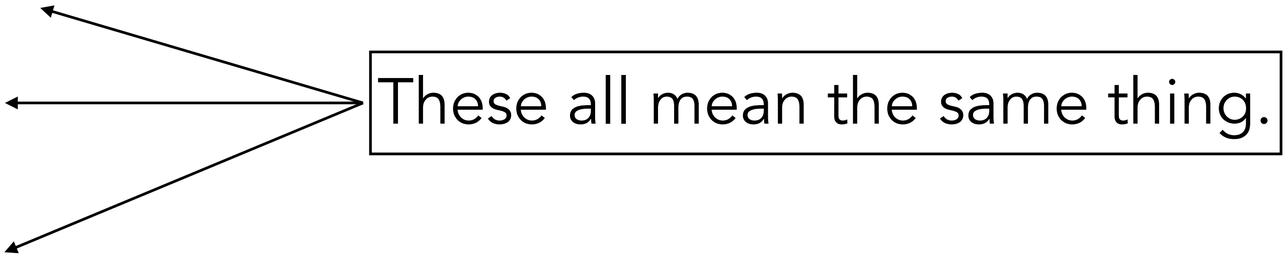
- How does the operating system protect itself from such threats?
- Combination of **hardware** and **software** protection
  - Hardware for interfaces at the granularity of instructions (e.g., setting the translation table base register; where the top-level page table is stored)
  - Software for interfaces at the granularity of system abstractions
    - Users are doing stuff with system calls, file system, etc… how do we ensure they have access to do what they want?

Privilege Levels

- What does privilege look like at the OS level?
  - “Privileged and non-privileged”
  - “Kernel mode and User mode”
  - “Supervisor and Normal”

Privilege Levels

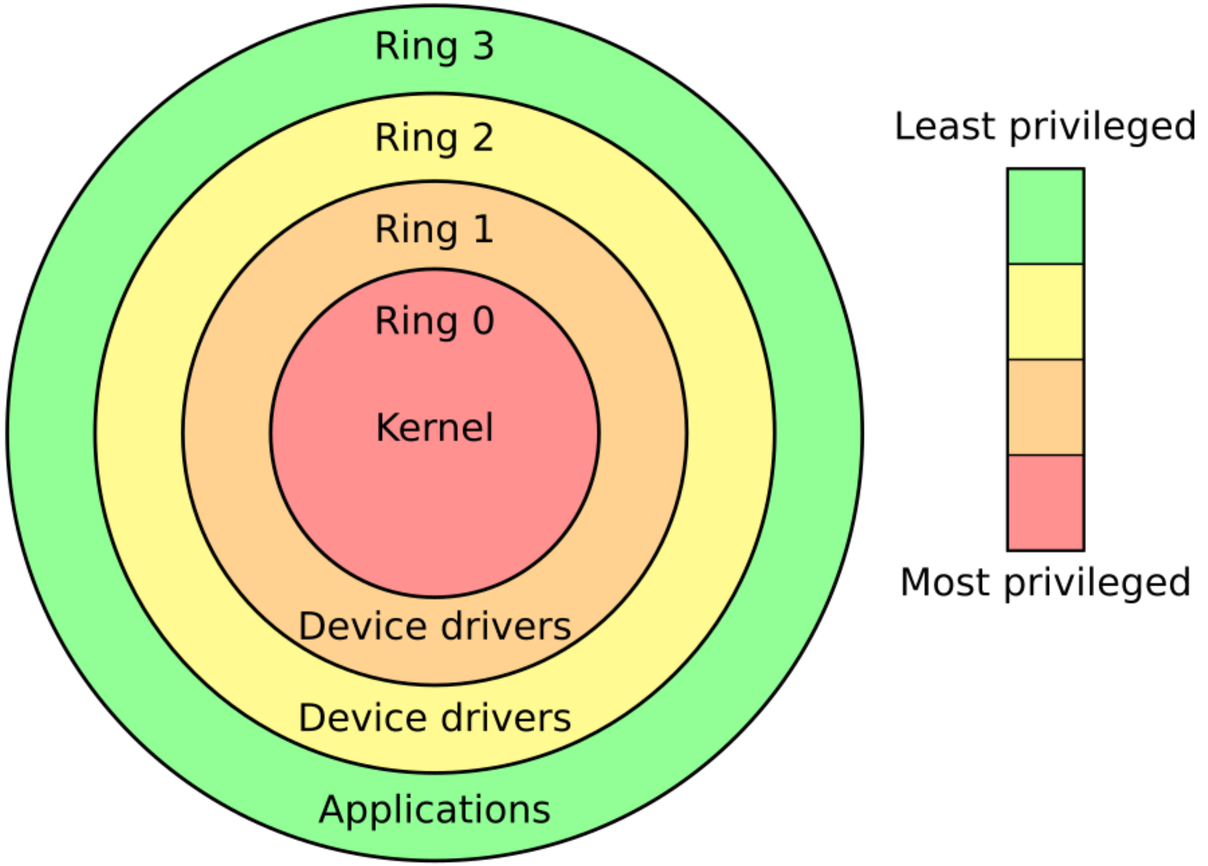
- What does privilege look like at the OS level?
  - “Privileged and non-privileged”
  - “Kernel mode and User mode”
  - “Supervisor and Normal”



- Processor is always operating at some privilege level
  - Held in a protected system register, varies by architecture

Intel Privilege Levels

- 4 rings (only Ring 0 and Ring 3 are typically used by OS)
  - Ring 0 is most privileged (kernel mode)
  - Ring 3 is least privileged (user mode)
- "Nothing is more privileged than the kernel" — wrong, turns out
  - Ring -1 (Hypervisor / Virtualization)
  - Ring -2 (System Management Mode)



https://en.wikipedia.org/wiki/Protection_ring

Changing Privilege in OSes

- In general, any process, program, entity can **give up their privilege freely** but cannot **gain arbitrary privileges**
- To enter a more privileged state, a process:
  - Prepares arguments, including where they they want to go
  - Executes a special instruction that initiates the transfer
- Remember **`int 0x80?`** This is a transfer of control… when you call a **kernel function**, you are instructing the hardware to change privilege state
  - Only a handful of ways for user mode program to enter kernel mode program… calls are completely media

System Calls

- Remember from 120: context switching is **costly**
  - And yet, userland programs needs a lot of help from the OS… (files, I/O, network, etc.)
  - Primarily interface with the OS through **syscalls;** how does this work?
- To make these fast, kernel's **virtual memory space** is mapped into every process, but made *inaccessible* when in user mode. How?
  - More protection bits: Unprivileged (UR, UW, UX) or Privileged (PR, PW, PX); these are also stored in each page table entry
  - When we say "kernel memory is at high address in x86" — this is what we mean

Kernel Privileges

- Poll: Does the kernel have access to usermode memory?

Kernel Privileges

- Poll: Does the kernel have access to usermode memory?
  - Yes! Mostly. Some finer details make this not 100% true, but in general, kernel can read / write any mapped pages from usermode
- So, kernel has to be super careful to keep track of whether it's working on kernel data or usermode data
  - Many classes of attacks are in usermode and trick the kernel into writing something from the user into kernel memory, leading to bad outcomes

A simple attack attempt

- `read()` system call
  - `ssize_t read(int fd, void *buf, size_t count);`
  - Basically, reads `count` bytes from the file specified by the file descriptor and writes it to `buf`
- What happens if an attacker (in usermode) sets buf to point to somewhere in kernel memory?

A simple attack attempt

- `read()` system call
  - `ssize_t read(int fd, void *buf, size_t count);`
  - Basically, reads `count` bytes from the file specified by the file descriptor and writes it to `buf`
- What happens if an attacker (in usermode) sets buf to point to somewhere in kernel memory?
  - Should fail. Why? Because kernel will check protections on `buf`, note that the call is coming from usermode, and block the read

A simple attack attempt

- `read()` system call
  - `ssize_t read(int fd, void *buf, size_t count);`
  - Basically, reads `count` bytes from the file specified by the file descriptor and writes it to `buf`
- What happens if an attacker (in usermode) sets buf to point to somewhere in kernel memory?
  - Should fail. Why? Because kernel will check protections on `buf`, note that the call is coming from usermode, and block the read

Kernel security

- This is hard to get right… so kernel developers have invented highly vetted functions that do just this
  - `copy_to_user()` and `copy_from_user()`
    - Functions that safely copy data between user and kernel buffers, checking for appropriate access in between
    - These are kind of like bouncers for kernel memory
  - Still, you could cause issues…
    - How many of you have ever set a variable to something, checked it later, and it's not what you expect?
    - Time of check vs. Time of use vulnerability… even messier with pointers

Kernel security

- This is hard to get right… so kernel developers have invented highly vetted functions that do just this
  - `copy_to_user()` and `copy_from_user()`
    - Functions that safely copy data between user and kernel buffers, checking for appropriate access in between
    - These are kind of like bouncers for kernel memory
  - Still, you could cause issues…
    - How many of you have ever set a variable to something, checked it later, and it's not what you expect?
    - Time of check vs. Time of use vulnerability… even messier with pointers

In sum, for OSes…

- Separate mechanisms for operating on usermode and kernel data
- Software
  - Keep track of usermode vs. kernel mode and be careful when deciding what to do (don't become a **confused deputy**)
- Hardware
  - Processor helps keep track of privilege levels for operations; transition into different levels of privilege is ultimately set in a protected register (on intel… other architectures may vary)

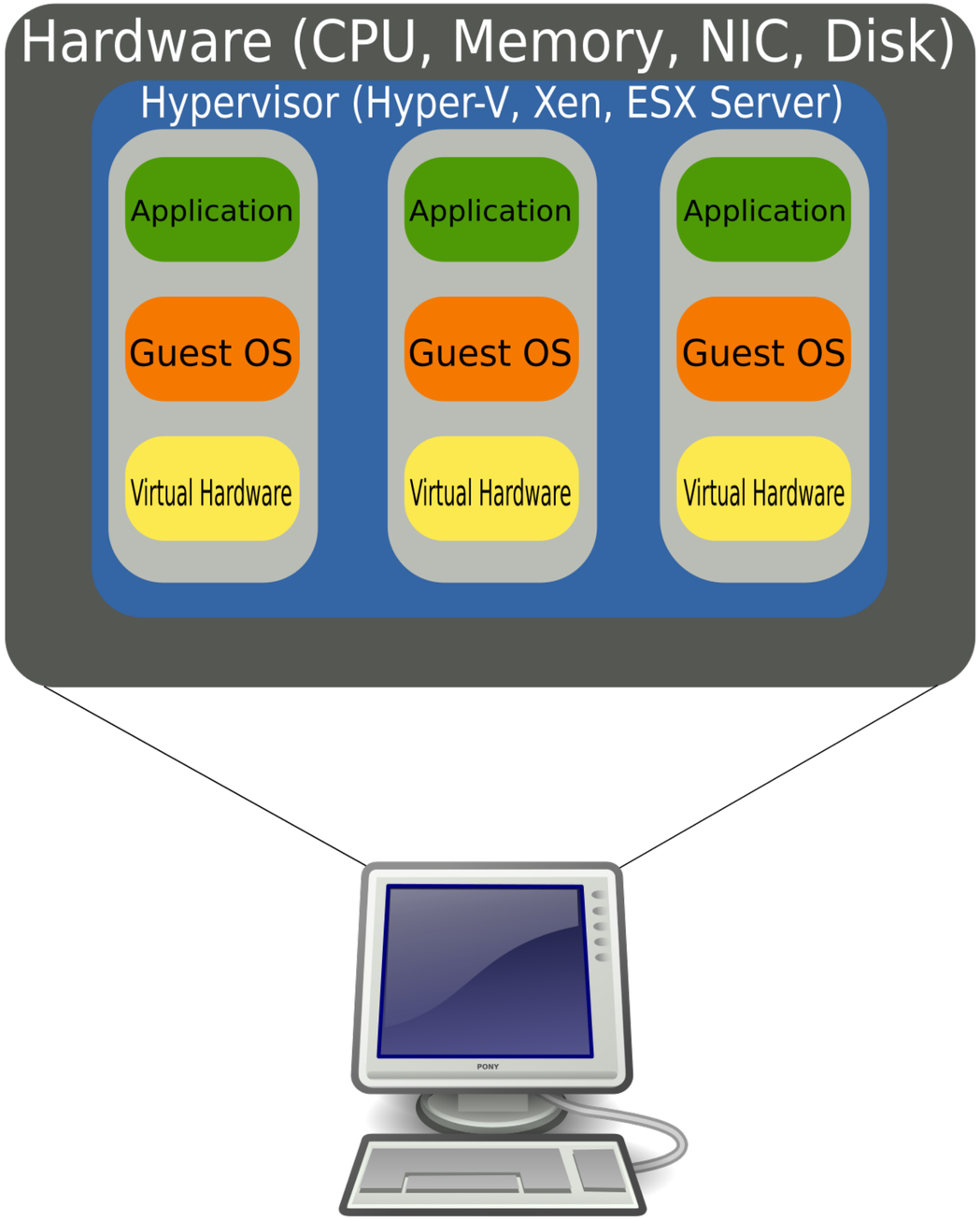
Virtualization

Virtual Machines

- What is a virtual machine?

Virtual Machines

- What is a virtual machine?
  - Deepak's version: "A computer running in your computer."
- Oh… but… how…
  - Specialized piece of software called a **hypervisor** implements VM environment and provides translations from the *guest* OS to the *host* OS
  - VirtualBox, UTM, VMWare… all hypervisors
- **The entire cloud is just VMs** (~hundreds of billions of dollar industry)



Virtual Machine Protections

- Your hypervisor can support **multiple virtual machines at the same time.** What protections are guaranteed?
  - Each virtual OS “thinks” its running on bare metal… it’s the hypervisor’s job to keep that illusion going
- You can think of a hypervisor as an OS for OSes… and often it needs special privileges to do that
  - Ring -1 in intel

Virtual Machine Details

Virtual Machine Details

- We won't cover them. You could spend full courses trying to understand them, and I won't pretend to know all the details.
- Just know…
  - There is now hardware support for virtualization since it's so popular (look into Intel VT-X and see if your chip supports it)
  - Some hypervisors can emulate *different architectures* (if you have an M* series Mac, this is what's happening for PA1 and PA2… bonkers!)
  - OSes should be totally protected from one another, ideally, they don't even know another OS is running on the same baremetal (and it's up to the hypervisor to prevent that)

In sum, for everything

- Operating systems have a **lot** going on.
- Process isolation
  - Hardware support (MMU)
  - Provide separate address spaces to different processes
  - Control modes of access to memory (i.e., R,W,X)
- User / Kernel Privilege Separation
  - Processor privilege modes used to limit access to sensitive instructions
  - Interfaces can cause lots of fun problems
- Virtual machines
  - Same idea, but add another layer of isolation

Next time

- Side channels!
  - Break the OS protection guarantees through *side channels*; leaking information to learn important stuff
  - In particular, Meltdown / Spectre (the most recent, impactful side channels circa 2018 that destroyed decades of architecture progress)